# COMPILER DESIGN AND CONSTRUCTION

By Evans Ombati Maoncha

LECTURE 3 AND 4

# Compiler

A compiler is a software that converts the source code to the object code. In other words, we can say that it converts the high-level language to machine/binary language. Moreover, it is necessary to perform this step to make the program executable. This is because the computer understands only binary language.

Some compilers convert the high-level language to an assembly language as an intermediate step. Whereas some others convert it directly to machine code. This process of converting the source code into machine code is called **compilation.**

The compiling process includes basic translation mechanisms and error detection. The compiler process goes through lexical, syntax, and semantic analysis at the front end and code generation and optimization at the back-end.


## Steps for Language processing systems

Before knowing about the concept of compilers, you first need to understand a few other tools which work with compilers.

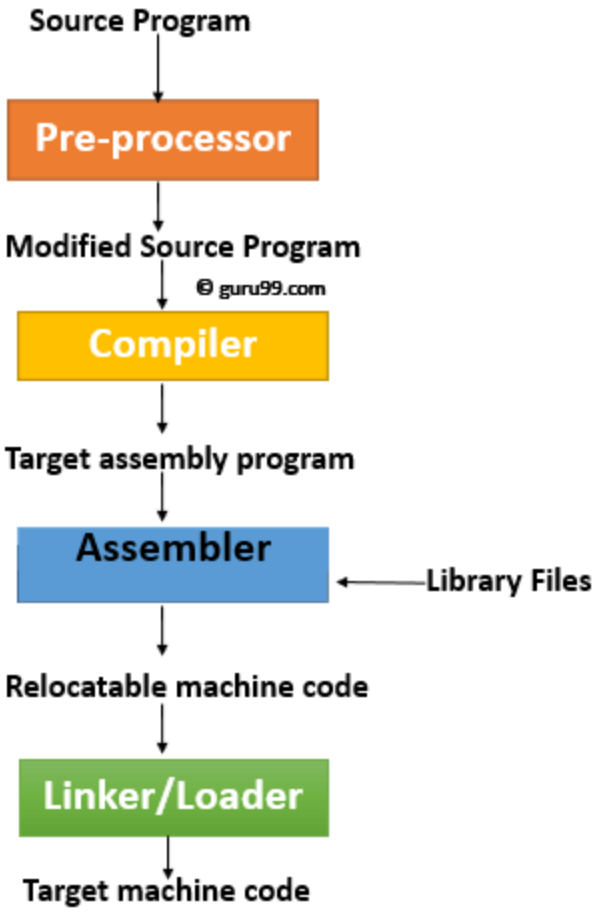



*Figure 1:Steps for Language processing systems*

- **Preprocessor**: The preprocessor is considered as a part of the Compiler. It is a tool which produces input for Compiler. It deals with macro processing, augmentation, language extension, etc.

- **Interpreter**: An interpreter is like Compiler which translates high-level language into low-level machine language. The main difference between both is that interpreter reads and transforms code line by line. Compiler reads the entire code at once and creates the machine code.

- **Assembler**: It translates assembly language code into machine understandable language. The output result of assembler is known as an object file which is a combination of machine instruction as well as the data required to store these instructions in memory.

- **Linker**: The linker helps you to link and merge various object files to create an executable file. All these files might have been compiled with separate assemblers. The main task of a linker is to search for called modules in a program and to find out the memory location where all modules are stored.

- **Loader**: The loader is a part of the OS, which performs the tasks of loading executable files into memory and run them. It also calculates the size of a program which creates additional memory space.

- **Cross-compiler**: A Cross compiler in compiler design is a platform which helps you to generate executable code.

- **Source-to-source Compiler**: Source to source compiler is a term used when the source code of one programming language is translated into the source of another language.

## Why use a Compiler?

- Compiler verifies entire program, so there are no syntax or semantic errors.
- The executable file is optimized by the compiler, so it is executes faster.
- Allows you to create internal structure in memory.
- There is no need to execute the program on the same machine it was built.
- Translate entire program in other language.
- Generate files on disk.
- Link the files into an executable format.
- Check for syntax errors and data types.
- Helps you to enhance your understanding of language semantics.
- Helps to handle language performance issues.
- Opportunity for a non-trivial programming project.
- The techniques used for constructing a compiler can be useful for other purposes as well.

## Application of Compilers

- Compiler design helps full implementation Of High-Level Programming Languages.
- Support optimization for Computer Architecture Parallelism.
- Design of New Memory Hierarchies of Machines.
- Widely used for Translating Programs.
- Used with other Software Productivity Tools.

## Analysis of a Source Program

We can analyze a source code in three main steps. Moreover, these steps are further divided into different phases. The three steps are:

### 1. Linear Analysis

Here, it reads the character of the code from left to right. The characters having a collective meaning are formed. We call these groups tokens.

### 2. Hierarchical Analysis

According to collective meaning, we divide the tokens hierarchically in a nested manner.

### 3. Semantic Analysis

In this step, we check if the components of the source code are appropriate in meaning.

## Phases/Structure of Compiler

The compilation process takes place in several phases. Moreover, for each step, the output of one step acts as the input for the next step. The phases/structure of the compilation process are is follows:

### 1. Lexical Analyzer

- It takes the high-level language source code as the input.
- It scans the characters of source code from left to right. Hence, the name scanner also.
- It groups the characters into lexemes. Lexemes are a group of characters which has some meaning.
- Each lexeme corresponds to form a token.
- It removes white spaces and comments.
- It checks and removes the lexical errors.

### 2. Syntax Analyzer

- ‘Parser’ is the other name for the syntax analyzer.
- The output of the lexical analyzer is its input.
- It checks for syntax errors in the source code.
- It does this by constructing a parse tree of all the tokens.
- For the syntax to be correct, the parse tree should be according to the rules of source code grammar.
- The grammar for such codes is context-free grammar.

### 3. Semantic Analyzer

- It verifies the parse tree of the syntax analyzer.

- It checks the validity of the code in terms of programming language. Like, compatibility of data types, declaration, and initialization of variables, etc.
- It also produces a verified parse tree. Furthermore, we also call this tree an annotated parse tree.
- It also performs flow checking, type checking, etc.

### 4. Intermediate Code Generator (ICG)

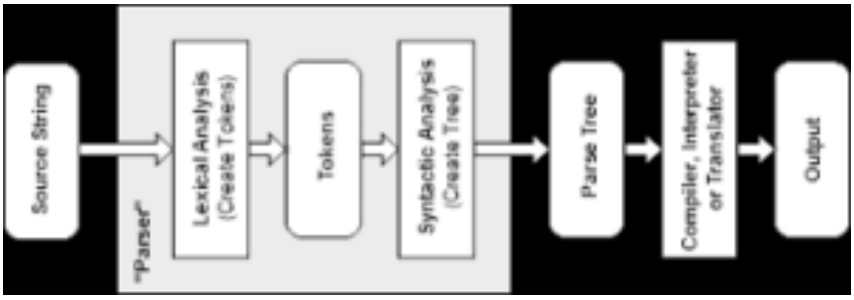
- It generates an intermediate code.
- This code is neither in high-level language nor in machine language. It is in an intermediate form.
- It is converted to machine language but, the last two phases are platform dependent.
- The intermediate code is the same for all the compilers. Further, we generate the machine code according to the platform.
- An example of an intermediate code is three address code.

### 5. Code Optimizer

- It optimizes the intermediate code.
- Its function is to convert the code so that it executes faster using fewer resources (CPU, memory).
- It removes any useless lines of code and rearranges the code.
- The meaning of the source code remains the same.

### 6. Target Code Generator

- Finally, it converts the optimized intermediate code into the machine code.
- This is the final stage of the compilation.
- The machine code which is produced is relocatable.




## Phases of Compiler

All these phases of a compiler divide into two sections:

### a) Front End

The phases of lexical analysis, syntax analysis, semantic analysis, and intermediate code generation comes under this category.

### b) Back End

While the other last two phases come under the back end.

## Types of Compilers

Following are the different types of Compiler:

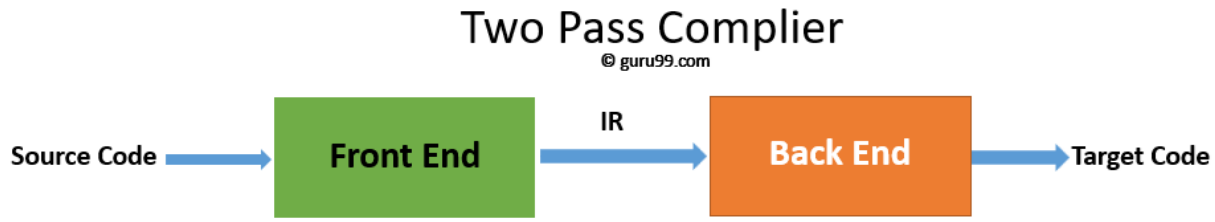
- Single Pass Compilers
- Two Pass Compilers
- Multipass Compilers

1. **Single Pass Compiler**


Single Pass Compiler

In single pass Compiler source code directly transforms into machine code. For example, Pascal language.

## 2. Two Pass Compiler



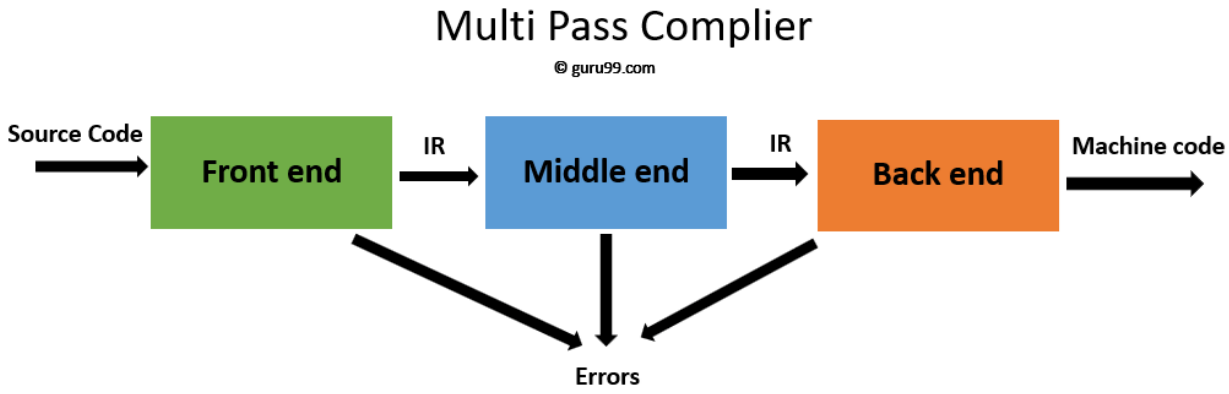

Two Pass Compiler

Two pass Compiler is divided into two sections, viz.

1. **Front end:** It maps legal code into Intermediate Representation (IR).
2. **Back end:** It maps IR onto the target machine

The Two pass compiler method also simplifies the retargeting process. It also allows multiple front ends.

## 3. Multipass Compilers




Multipass Compilers

The multipass compiler processes the source code or syntax tree of a program several times. It divides a large program into multiple small programs and process them. It develops multiple intermediate codes. All of these multipass take the output of the previous phase as an input. So it requires less memory. It is also known as 'Wide Compiler'.

**Other Classification of compilers are: -**

### 4. Cross Compilers

They produce an executable machine code for a platform but, this platform is not the one on which the compiler is running.

### 5. Bootstrap Compilers

These compilers are written in a programming language that they have to compile.

### 6. Source to source/transcompiler

These compilers convert the source code of one programming language to the source code of another programming language.

### 7. Decompiler

Basically, it is not a compiler. It is just the reverse of the compiler. It converts the machine code into high-level language.

## Features of a Compiler

The features are as follows:

- Compilation speed.
- The correctness of machine code.
- The meaning of code should not change.
- Speed of machine code.
- Good error detection.
- Checking the code correctly according to grammar.

## Uses/Application of Compilers

- Helps to make the code independent of the platform.
- Makes the code free of syntax and semantic errors.
- Generate executable files of code.
- Translates the code from one language to another.

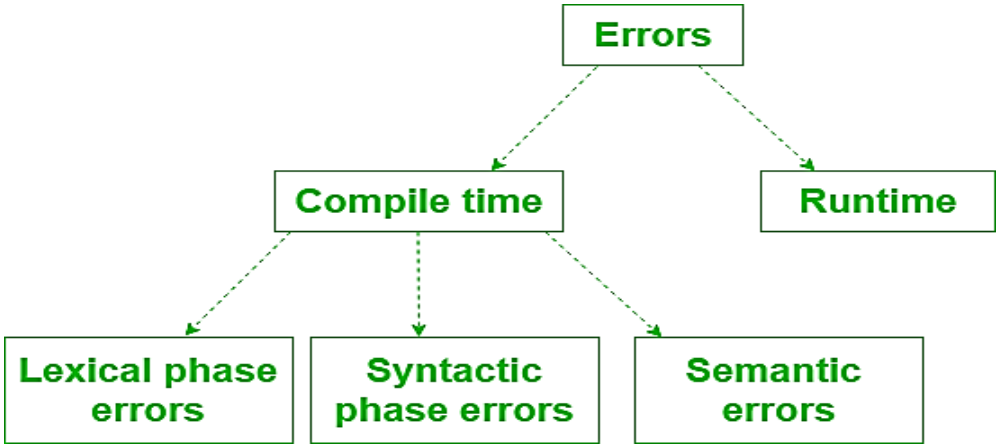
# Error detection and Recovery in Compiler

In this phase of compilation, all possible errors made by the user are detected and reported to the user in form of error messages. This process of locating errors and reporting them to users is called the **Error Handling process**.

## Functions of an Error handler

- Detection
- Reporting
- Recovery

## Classification of Errors




Compile-time errors are of three types: -

### Lexical phase errors

These errors are detected during the lexical analysis phase. Typical lexical errors are:

- Exceeding length of identifier or numeric constants.
- The appearance of illegal characters
- Unmatched string

Example 1 : **printf("Hello World");$**
This is a lexical error since an illegal character $ appears at the end of statement.

Example 2 : **This is a comment */**
This is an lexical error since end of comment is present but beginning is not present.

**Error recovery:**

*Panic Mode Recovery*

In this method, successive characters from the input are removed one at a time until a designated set of synchronizing tokens is found. Synchronizing tokens are delimiters such as; or }

- The advantage is that it is easy to implement and guarantees not to go into an infinite loop
- The disadvantage is that a considerable amount of input is skipped without checking it for additional errors

**Syntactic phase errors**

These errors are detected during the syntax analysis phase. Typical syntax errors are:

- Errors in structure
- Missing operator
- Misspelled keywords
- Unbalanced parenthesis

**Example:** switch(ch)

```
{
  .......
  .......
}
```

The keyword **switch** is incorrectly written as a switch. Hence, an **"Unidentified keyword/identifier"** error occurs.

**Error recovery:**

1. **Panic Mode Recovery**
    - In this method, successive characters from the input are removed one at a time until a designated set of synchronizing tokens is found. Synchronizing tokens are deli-meters such as; or }
    - The advantage is that it's easy to implement and guarantees not to go into an infinite loop
    - The disadvantage is that a considerable amount of input is skipped without checking it for additional errors
2. **Statement Mode recovery**
    - In this method, when a parser encounters an error, it performs the necessary correction on the remaining input so that the rest of the input statement allows the parser to parse ahead.

- The correction can be deletion of extra semicolons, replacing the comma with semicolons, or inserting a missing semicolon.
- While performing correction, utmost care should be taken for not going in an infinite loop.
- A disadvantage is that it finds it difficult to handle situations where the actual error occurred before pointing of detection.

3. **Error production**
   - If a user has knowledge of common errors that can be encountered then, these errors can be incorporated by augmenting the grammar with error productions that generate erroneous constructs.
   - If this is used then, during parsing appropriate error messages can be generated and parsing can be continued.
   - The disadvantage is that it's difficult to maintain.
4. **Global Correction**
   - The parser examines the whole program and tries to find out the closest match for it which is error-free.
   - The closest match program has less number of insertions, deletions, and changes of tokens to recover from erroneous input.
   - Due to high time and space complexity, this method is not implemented practically.

**Semantic errors**

These errors are detected during the semantic analysis phase. Typical semantic errors are

- Incompatible type of operands
- Undeclared variables
- Not matching of actual arguments with a formal one

**Example :** int a[10], b;

```
.......
.......
a = b;
```

It generates a semantic error because of an incompatible type of a and b.

**Error recovery**

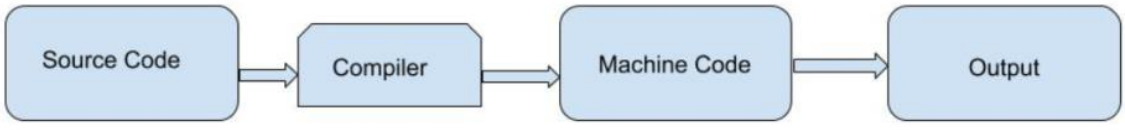
- If the error **"Undeclared Identifier"** is encountered then, to recover from this a symbol table entry for the corresponding identifier is made.
- If data types of two operands are incompatible then, automatic type conversion is done by the compiler.

## Difference Between Compiler and Interpreter

A compiler checks the whole program at once. It displays all the errors at a place once the whole program is checked. On the other hand, an interpreter checks the program line by line. If an error is detected the execution stops.

# Interpreter

An interpreter is a program that directly executes the instructions in a high-level language, without converting it into machine code. In programming, we can execute a program in two ways. Firstly, through compilation and secondly, through an interpreter. The common way is to use a compiler.




How Compiler Works


How Interpreter Works

## Difference Between Compilers and Interpreters

| Sr.No | Compilers | Interpreters |
|---|---|---|
| 1. | It converts the whole program into machine code at once. | It translates only one statement at a time. |

| | | |
|---|---|---|
| 2. | It takes more time to analyze the source code. In other words, compile time is more. However, the overall execution time is less. | It comparatively takes less time to analyze the source code. In other words, compile time is less. However, the overall execution time is more. |
| 3. | It generates an intermediate object code. Therefore, more memory is used. | It does not generate any intermediate object code. Hence it is memory efficient. |
| 4. | The whole program is compiled and then it shows all the errors together. Therefore, debugging is difficult. | It stops the compilation if any error occurs. Hence, debugging is easier. |
| 5. | Programming languages like C, C++, Java, etc use compiler. | Programming languages like Python, Ruby, PHP, etc. use an interpreter. These interpreted languages are also called **scripting languages**. |

# Summary

- A compiler is a computer program which helps you transform source code written in a high-level language into low-level machine language.
- Correctness, speed of compilation, preserve the correct the meaning of the code are some important features of compiler design.
- Compilers are divided into three parts 1) Single Pass Compilers 2)Two Pass Compilers, and 3) Multipass Compilers.
- The “compiler” was word first used in the early 1950s by Grace Murray Hopper.
- Steps for Language processing system are: Preprocessor, Interpreter, Assembler, Linker/Loader.
- Important compiler construction tools are 1) Scanner generators, 2)Syntax-3) directed translation engines, 4) Parser generators, 5) Automatic code generators.
- The main task of the compiler is to verify the entire program, so there are no syntax or semantic errors.

# Frequently Asked Questions (FAQs)

Q1. What is a compiler?

A1. It is software that converts the source code into machine code. The process is called compilation.

Q2. What are the phases/structure of a compiler?

A2. The phases are:

- lexical analyzer
- syntax analyzer
- semantic analyzer
- intermediate code generator
- code optimizer
- target code generator

Q3. What is a symbol table?

A3. It helps to find the names of identifiers easily. It consists of identifiers and their types.

Q4. What is the difference between compiler and interpreter?

A4. The compiler checks the code as a whole whereas, the interpreter checks it line by line.

Q5. What is a decompiler?

A5. It converts machine code to the source code. It is the reverse of the compiler.

Q6. What Is Linear Analysis?

A6. Linear analysis is one in which the stream of characters making up the source program is read from left to right and grouped into tokens that are sequences of characters having a collective meaning. Also called lexical analysis or scanning.